A General and Parallel Platform for Mining Co-Movement Patterns over Large-scale Trajectories

ABSTRACT

1. INTRODUCTION

The prevalence of positioning devices has drastically boosted the scale and spectrum of trajectory collection to an unprecedented level. Tremendous amounts of trajectories, in the form of sequenced spatial-temporal records, are continually generated from animal telemetry chips, vehicle GPSs and wearable devices. Data analysis on large-scale trajectories benefits a wide range of applications and services, including traffic planning [1], animal analysis [2], and social recommendations [3], to name just a few.

A crucial task of data analysis on top of trajectories is to discover co-moving patterns. A *co-movement* pattern [4] refers to a group of objects traveling together for a certain period of time and the group is normally determined by spatial proximity. A pattern is prominent if the size of the group exceeds M and the length of the duration exceeds K, where M and K are parameters specified by users. Rooted from such basic definition and driven by different mining applications, there are a bunch of variants of co-movement patterns that have been developed with more advanced constraints.

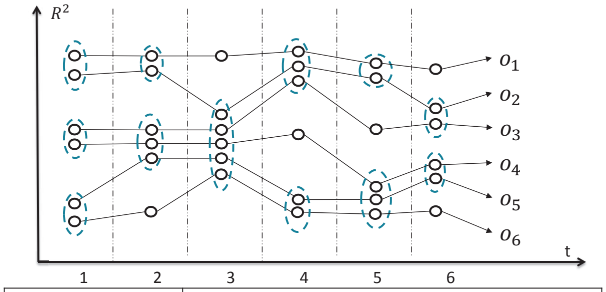
Table 1 summarizes several popular co-moving pattern s with different constraints in the attributes of clustering in spatial proximity, consecutiveness in temporal duration and computational complexity. In particular, the *flock* [5] and the *group* [6] patterns require all the objects in a group to be enclosed by a disk with radius r; whereas the *convoy* [7], the *swarm* [8] and the *platoon* [9] patterns resort to density-based spatial clustering. In the temporal dimension, the *flock* [5] and the *convoy* [7] require all the timestamps of each detected spatial group to be consecutive, which is referred to as *global consecutiveness*; whereas the *swarm* [8] does not impose any restriction. The *group* [6] and the *platoon* [9] adopt a compromised manner by allowing arbitrary gaps between the consecutive segments, which is called *local consecutiveness*. They introduce a parameter L to control the minimum length of each local consecutive segment.

Patterns	Proximity	Consecutiveness	Time Complexity
flock [10]	disk-based	global	$O(\lvert\mathbb{O}\rvert\lvert\mathbb{T}\rvert(M + log(\lvert\mathbb{O}\rvert))$
convoy [7]	density-based	global	$O(\lvert\mathbb{O}\rvert^2 + \lvert\mathbb{O}\rvert\lvert\mathbb{T}\rvert)$
swarm [8]	density-based	-	$O(2^{\lvert\mathbb{O}\rvert}\lvert\mathbb{O}\rvert\lvert\mathbb{T}\rvert)$
group [6]	disk-based	local	$O(\lvert\mathbb{O}\rvert^2\lvert\mathbb{T}\rvert)$
platoon [9]	density-based	local	$O(2^{\lvert\mathbb{O}\rvert}\lvert\mathbb{O}\rvert\lvert\mathbb{T}\rvert)$

Table 1: Constraints and complexity of co-movement patterns. The time complexity indicates the performance in the worst case, where $|\mathbb{O}|$ is the total number of objects and $|\mathbb{T}|$ is the number of descritized timestamps.

Figure 1 is an example to demonstrate the concepts of various co-movement patterns. The trajectory database consists of six moving objects and the temporal dimension is discretized into six snapshots. In each snapshot, we treat the clustering methods as a black-box and assume that they generate the same clusters. Objects in proximity are grouped in the dotted circles. As aforementioned, there are three parameters to determine the co-movement patterns and the default settings in this example are $M = 2$, $K = 3$ and $L = 2$. Both the *flock* and the *convoy* require the spatial clusters to last for at least K consecutive timestamps. Hence,$\{o_3, o_4\}$ and $\{o_5, o_6\}$ remains the only two candidates matching the patterns. The *swarm* relaxes the pattern matching by discarding the temporal consecutiveness constraint. Thus, it generates many more candidates than the *flock* and the *convoy*. The *group* and the *platoon* add another constraint on local consecutiveness to retain meaningful patterns. For instance, $\{o_1, o_2 : 1, 2, 4, 5\}$ is a pattern matching local consecutiveness because timestamps $\{1, 2\}$ and $\{4, 5\}$ are two segments with length no smaller than $L = 2$. The difference between the *group* and the *platoon* is that the *platoon* has an additional parameter K to specify the minimum number of snapshots for the spatial clusters. This explains why $\{o_3, o_4, o_5 : 2, 3\}$ is a *group* pattern but not a *platoon* pattern.

As can be seen, there are various co-movement patterns requested by different applications and it is cumbersome to design a tailored solution for each type. In addition, despite the generality of the *platoon* (i.e., it can be reduced to other types of patterns via proper parameter settings), it suffers from the so-called *loose-connection* anomaly. We use two objects o_1 and o_2 in Figure 2 as an example to illustrate the scenario. These two objects form a *platoon* pattern in timestamps $\{1, 2, 3, 102, 103, 104\}$. However, the two consecutive segments are 98 timestamps apart, resulting in a false positive co-movement pattern. In reality, such an anomaly may be caused by the periodic movements of unrelated ob-



Flock(M=2, K=3)	$\{o_3, o_4$: 1,2,3$\}$ $\{o_5, o_6$: 3,4,5$\}$
Convoy(M=2, K=3)	$\{o_3, o_4$: 1,2,3$\}$ $\{o_5, o_6$: 3,4,5$\}$
Group(M=2, L=2)	$\{o_1, o_2$: 1,2,4,5$\}$ $\{o_3, o_4, o_5$: 2,3$\}$ $\{o_5, o_6$: 3,4,5$\}$
Platoon (M=2,K=3,L=2)	$\{o_1, o_2$: 1,2,4,5$\}$ $\{o_3, o_4$: 1,2,3$\}$ $\{o_4, o_5$: 2,3,5,6$\}$ $\{o_5, o_6$: 3,4,5$\}$
Swarm (M=2,K=3)	$\{o_1, o_2$: 1,2,4,5$\}$ $\{o_3, o_4$: 1,2,3$\}$ $\{o_4, o_5$: 2,3,5,6$\}$ $\{o_5, o_6$: 1,3,4,5$\}$ $\{o_2, o_3$: 3,4,6$\}$

Figure 1: Trajectories and co-movement patterns; The example consists of six trajectories across six snapshots. Objects in spatial clusters are enclosed by dotted circles. M is the minimum cluster cardinality; K denotes the minimum number of snapshots for the occurrence of a spatial cluster; and L denotes the minimum length for local consecutiveness.

jects, such as vehicles stopping at the same petrol station or animals pausing at the same water source. Unfortunately, none of the existing patterns have directly addressed this anomaly.

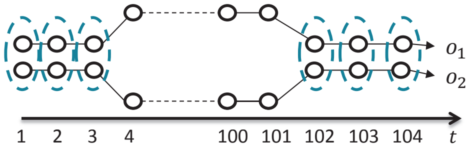
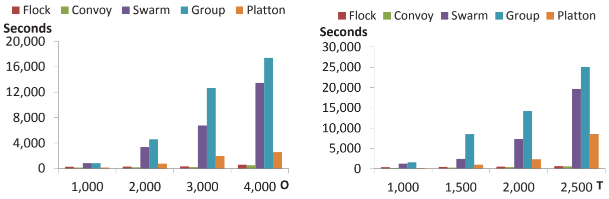


Figure 2: *Loose-connection* anomaly. Even though $\{o_1, o_2 : 1, 2, 3, 102, 103, 104\}$ is considered as a valid *platoon* pattern, it is highly probable that these two objects are not related as the two consecutive segments are 98 timestamps apart.

The other issue with existing methods is that they are built on top of centralized indexes which may not be scalable. Table 1 shows their theoretical complexities in the worst cases and the largest real dataset ever evaluated in previous studies is up to million-scale points collected from hundreds of moving objects. In practice, the dataset is of much higher scale and the scalability of existing methods is left unknown. Thus, we conduct an experimental evaluation with 4000 objects moving for 2500 timestamps to examine the scalability. Results in Figure 3 show that their performances degrade dramatically as the dataset scales up. For instance, the detection time of *group* drops twenty times as the number of objects grows from *1k* to *4k*. Similarly, the performance of *swarm* drops over fifteen times as the number of snapshots grows from *1k* to *2.5k*. These observations imply that existing methods are not scalable to support large-scale trajectory databases.

Therefore, our primary contributions in this paper are to close these two gaps. First, we propose the *general co-movement pattern* (GCMP) which models various co-moment patterns in a unified way and can avoid the *loose-connection* anomaly. In GCMP, we introduce a new gap parameter G to pose a constraint on the temporal gap between two consecutive segments. By setting a feasible G, the loose-connection anomaly can be avoided. In addition, our GCMP is both general and expressive. It can be reduced to any of the previous patterns by customizing the parameters.



(a) $1k$ timestamps (b) $1k$ objects

Figure 3: Performance measures on existing co-movement patterns. A sampled Geolife data set is used with up two 2.4 million data points. Default parameters are $M = 10$ $K = 20$ $L = 10$.

Second, we investigate deploying our GCMP detector on MapReduce platforms (such as Hadoop and Spark) to tackle the scalability issue. Our technical contributions are threefold. First, we replicate the snapshots in multiple data chunks to support efficient parallel processing. Second, we devise a novel *Star Partition and Mining* (SPM) algorithm as a fine-granularity partitioning strategy to achieve workload balance. For each star, an Apriori method is adopted to mine the co-movement patterns. Third, we leverage the *temporal monotonicity* property of GCMP to design several optimization techniques including *edge-simplification* to prune initial candidates, *temporal apriori pruning* and *forward closure checking* to reduce the number of enumerated candidates in the Apriori algorithm.

We conduct a set of extensive experiments on three large-scaled real datasets with hundreds of millions temporal points. The results show that our parallel scheme efficiently supports GCMP mining in large datasets. In particular, with near 200 million trajectory points, our scheme runs within 10 minutes using 135 cores. Whereas centralized solution takes near 7 hours for 1 million trajectory points. Moreover, our optimized SPM methods achieves upto XXX times efficiency as compared to the baseline algorithm with near linear scalability.

The rest of our paper is organized as follows: Section 2 summarizes the relevant literature on trajectory pattern mining. Section 3 states the problem definition of our general co-movement pattern mining. Section 4 provides a baseline solution. An advanced solution named *star partition and mining* is presented in Section 5. Section 6 conducts extensive experiments to verify the efficiency of our system. Finally Section 7 concludes the paper.

2. RELATED WORKS

The *co-movement patterns* in literature consist of five members, namely *group* [6], *flock* [10], *convoy* [7], *swarm* [8] and *platoon* [9]. We have demonstrated the semantics of these patterns in Table 1 and Figure 1. In this section, we focus on comparing the techniques used in these works. For more trajectory patterns other than *co-movement patterns*, interested readers may move to [11] for a comprehensive survey.

2.1 Flock and Convoy

The difference between *flock* and *convoy* lies in the object clustering methods. In *flock* objects are clustered based on their distance. Specifically, the objects in the same cluster needs to have a pair-wised distance less than *min_dist*. This essentially requires the objects to be within a disk-region of delimiter less than *min_dist*. In contrast, *convoy* cluster the objects using density-based clustering [12]. Technically, *flock* utilizes a m^{th}-order Voronoi diagram [13] to detect whether a subset of object with size greater than m stays in a disk-region. *Convoy* employs a trajectory simplification [14] technique to boost pairwise distance computations in the density-based clustering. After clustering, both *flock* and *convoy* use a line-sweep method to scan each snapshots. During the scan, the object group appears in consecutive timestamps is detected. Meanwhile, the object groups that do not match the consecutive constraint are pruned. However, such a method faces high complexity issues when supporting other patterns. For instance, in *swarm*, the candidate set during the line-sweep grows exponentially, and many candidates can only be pruned after the entire snapshots are scanned.

2.2 Group, Swarm and Platoon

Different from *flock* and *convoy*, all the *group*,*swarm* and *platoon* patterns have more constraints on the pattern duration. Therefore, their techniques of mining are of the same skeleton. The main idea of mining is to grow object set from an empty set in a depth-first manner. During the growth, various pruning techniques are provided to prune unnecessary branches. *Group* pattern uses the Apriori property among patterns to facilitate the pruning. *Swarm* adapts two more pruning rules called backward pruning and forward pruning. *Platoon* further adapts a prefix table structure to guide the depth-first search. As shown by Li et.al. [9], *platoon* outperforms other two methods in efficiency. However, the three patterns are not able to directly discover the general co-movement pattern. Furthermore, their pruning rules heavily rely on the depth-first search nature, which lost its efficiency in the parallel scenario.

2.3 MaReduce Framework

MapReduce (MR) was formally proposed by Dean et.al. [15] and has subsequently implemented by many open source systems. Those systems provide handy APIs with fault tolerances and are popularly used as large-scale data processing platforms. In simple words, there are two conceptual types of computing nodes in MR, namely the *mapper*s and the *reducer*s. The execution of a MR algorithm consists of three major steps: First, input data are partitioned and read by a *map* function on each mapper. Then, mappers emit key-value pairs which are *shuffle*d over the network to reducers. Lastly, reducers process the received data using a *reduce* function.

3. DEFINITIONS

Let $\mathbb{O} = \{o_1, o_2, ..., o_n\}$ be the set of objects and $\mathbb{T} = (1, 2, ..., m)$ be the discretized temporal dimension. A time sequence T is defined as a ordered subset of $\mathbb{T}$. Let $T[i]$ be the i-th entry of the sequence T and i starts from 1. We allow the traditional set operators such as *membership*, *union*, *intersect*, *difference*, *candinality*, *min*, *max* and *range* on sequences without changing their semantics. Since sequences are ordered, we distinguish the concept of *subset* and *subsequence*. Given two sequence T_1 and T_2, T_1 is a *subset* of T_2, if $\forall t \in T_1$ implies $t \in T_2$. T_1 is a *subsequence* of T_2, if $\exists i \in (0, .., |T_2|), \forall j \in (1, .., |T_1|), T_1[j] = T_2[i + j]$. It is notable that a subsequence is always a subset, but not vice versa. We use $T[i : j]$ to denote the subsequence of T from i-th entry to j-th entry inclusively.

We say a sequence T is consecutive if $\forall i \in (1, ..., |T| - 1), T[i + 1] = T[i] + 1$. We refer a consecutive portion of T as a *segment*. It is obvious that any time sequence T can be decomposed into segments and we say T is *L-consecutive* [9] if the length of every segment is no smaller than L.

As illustrated in Figure 2, patterns adopting the notion of L-consecutiveness (e.g., *platoon* and *group*) still suffer from the *loose connection* problem. To avoid such an anomaly without losing pattern generality, we introduce a parameter G to control the gaps between timestamps in a pattern. Formally, a G-connected time sequence is defined as follows:

Definition 1 (G-connected). *A time sequence T is G-connected if the gap between any of its neighboring timestamps is no greater than G. That is $\forall i \in (1, ..., |T| - 1), T[i + 1] - T[i] \leq G$.*

We take $T = (1, 2, 3, 5, 6)$ as an example, which can be decomposed into two segments $(1, 2, 3)$ and $(5, 6)$. T is not 3-consecutive since the length of $(5, 6)$ is 2. Thus, it is safe to say either T is 1-consecutive or 2-consecutive. On the other hand, T is 2-connected since the maximum gap between its neighboring time stamps is $5 - 3 = 2$. It is worth noting that T is not 1-connected because the gap between $T[3]$ and $T[4]$ is 2 (i.e., 5-3=2).

Given a trajectory database discretized into snapshots, we can conduct a clustering method, either disk-based or density-based, to identify groups with spatial proximity. Let T be the set of timestamps in which a group of objects O are clustered. We are ready to define a more general co-movement pattern:

Definition 2 (General Co-Movement Pattern). *A general co-movement pattern finds a set of objects O satisfying the following five constraints: 1) closeness: the objects in O belong to the same cluster in the timestamps of T; 2) significance: $|O| \geq M$; 3) duration: $|T| \geq K$; 4) consecutiveness: T is L-consecutive; and 5) connection: T is G-connected.*

There are four parameters in our general co-movement pattern, including object constraint M and temporal constraints K, L, G. By customizing these parameters, our pattern can express other patterns proposed in previous literature, as illustrated in Table 2. In particular, by setting $G = |\mathbb{T}|$, we achieve the *platoon* pattern. By setting $G = |\mathbb{T}|, L = 1$, we achieve the *swarm* pattern. By setting $G = |\mathbb{T}|$, $M = 2$, $K = 1$, we gain the *group* pattern. Finally by setting $G = 1$, we achieve the *convoy* and *flock* patterns. In addition to the flexibility of representing other existing patterns, our GCMP is able to avoid the *loose connection* anomaly by tuning the parameter G. It is notable that GCMP cannot be modeled by existing patterns.

It is also observable that the number of patterns in GCMP could be exponential under some parameter settings (i.e., when expressing *swarm*). In particular, given a parameter M, if a pattern P is valid, then any subset of P with size M is also a valid pattern. This results in additional $\Sigma_{M \geq i \geq |P.O|} \binom{|P.O|}{i}$ patterns, which is clearly overwhelming

Pattern	M	K	L	G	Clustering
Group	2	1	2	$\lvert\mathbb{T}\rvert$	Disk-based
Flock	$\cdot$	$\cdot$	K	1	Disk-based
Convoy	$\cdot$	$\cdot$	K	1	Density-based
Swarm	$\cdot$	$\cdot$	1	$\lvert\mathbb{T}\rvert$	Density-based
Platoon	$\cdot$	$\cdot$	$\cdot$	$\lvert\mathbb{T}\rvert$	Density-based

Table 2: Expressing other patterns using GCMP. $\cdot$ indicates a user specified value. M represents the object *size* constraint. K represents the *duration* constraint. L represents the *consecutiveness* constraint. G represents the *connection* constraint.

and redundant. For all these patterns, output P is sufficient. Therefore, we define the *Closed General Co-Movement Pattern* as follows:

Definition 3 (Closed General Co-Movement Pattern)**.** *A general co-moving pattern $P = \langle O : T\rangle$ is closed if and only if there does not exist another general co-moving pattern P' s.t. $P.O \subseteq P'.O$.*

For example, let $M = 2, K = 2, , L = 1, G = 1$. In Figure 1, the pattern $P_1 = \{o_3, o_4 : 1, 2, 3\}$ is not a closed pattern. This is because $P_2 = \{o_3, o_4, o_5 : 2, 3\}$ is a closed pattern since $P_2.O \supset P_1.O$. The closed pattern avoids outputting duplicate information, thus making the result patterns more compact.

Our definition of GCMP is independent of the clustering method. Users can apply different clustering methods to facilitate different application needs. We currently expose both disc-region based clustering and DBSCAN as options to the users.

In summary, the goal of this paper is to present a parallel solution for discovering closed GCMP from large-scale trajectory datasets.

Before we move on to the algorithmic part, we list the notations that are used in the following sections.

Symbols	Meanings
S_t	snapshot of objects at time t
M	object size constraint
K	duration constraint
L	consecutiveness constraint
G	connection constraint
$P = (O : T)$	pattern with object set O, time sequence T
η	replication factor in TRM
$C_t(o)$	the cluster of object o at time t
S_t	the set of clusters at time t
λ_i	the partition with snapshots $S_t, .., S_{t+\eta-1}$
Sr_i	the star of object i

Table 3: Symbols and notions used

4. BASELINE: TEMPORAL REPLICATION AND PARALLEL MINING

In this section, we propose a baseline solution that resorts to MapReduce (MR) as a general, parallel and scalable paradigm for GCMP pattern mining. The framework, named *temporal replication and parallel mining* (TRPM), is illustrated in Figure 4. There are two cycles of map-reduce jobs connected in a pipeline manner. The first cycle deals with spatial clustering in each snapshot, which can be seen as a preprocessing step for the subsequent pattern mining. In particular, the timestamp is treated as the key in the map phrase and objects within the same snapshot are clustered (DBSCAN or disk-based clustering) in the reduce phrase. Finally, the reducers output clusters of objects in each snapshot, represented by a list of S_t, where t is the timestamp and S_t is a set of clustered objects at snapshot t.

Our focus in this paper is the second map-reduce cycle of parallel mining, which essentially consists of two key questions to solve. The first is how to employ effective data partitioning such that the mining can be conducted independently; and the second is how to efficiently mine the valid patterns within each partition.

It is obvious that we cannot simply split the trajectory database into disjoint partitions because a GCMP pattern requires L-consecutiveness and the corresponding segments may cross multiple partitions. Our strategy is to use data replication to enable parallel mining. Each snapshot will replicate its clusters to $\eta - 1$ preceding snapshots. In other words, the partition for the snapshot S_t contains clusters in S_t, $S_{t+1} \dots, S_{t+\eta-1}$. Determining a proper η is critical in ensuring the correctness and efficiency of TRPM. If η is too small, certain cross-partition patterns may be missed. If η is set too large, expensive network communication and CPU processing costs would be incurred in the map and reduce phrases respectively.

Before moving on to find the value of η, we first use $\rho_{G,L,K}(T)$ to define the *min-spanned valid subsequence* of a sequence T. In simple words, $\rho_{G,L,K}(T)$ is the subsequence of T with the smallest range and is valid wrt. K, L, G. To guarantee that no valid patterns are missing, η needs to be greater than the range of $\rho_{G,L,K}(T)$ for any valid sequence T. To see this, consider an arbitrary valid sequence T. Since each partition contains η snapshots, if $\eta \geq \text{range}(\rho_{G,L,K}(T))$ holds, $\rho_{G,L,K}(T)$ must be completely located in one partition. Therefore, the patterns associated with T would not be missing. Formally, η can be computed as:

$$\eta = \max\{\text{range}(\rho_{G,L,K}(T)) | T \text{ is valid wrt. } G, L, K\} \quad (1)$$

Directly compute such η is challenging. However, we provide the following theorem which states a bounded estimation on η:

Theorem 1. *Let $\eta = (\lceil \frac{K}{L} \rceil - 1) * (G-1) + K + L - 1$. Then it guarantees the completeness of the partition. Further: let η^* be the optimal value for a given G, L, K, then the following holds: $\eta \leq \eta^* + L - 1$.*

Proof. See Appendix. □

Based on the above theorem, during TRPM, every consecutive η snapshots form a partition. In particular, for every snapshot S_t, there is a partition $\lambda_t = \{S_t, ..., S_{t+\eta-1}\}$. With this partition strategy, when discovering GCMPS in λ_t, we only need to keep the pattern candidates whose object set are in S_t. This motivates to design an efficient *line-sweep* mining (LSM) method for each partition. The intuition of LSM is to sequentially scan all snapshots. During scanning, a set of pattern candidates is maintained. When all snapshots are scanned, the remaining candidates are the true patterns. The detail of LSM is presented in Algorithm 1. A candidate set C is maintained throughout the algorithm(line 1).

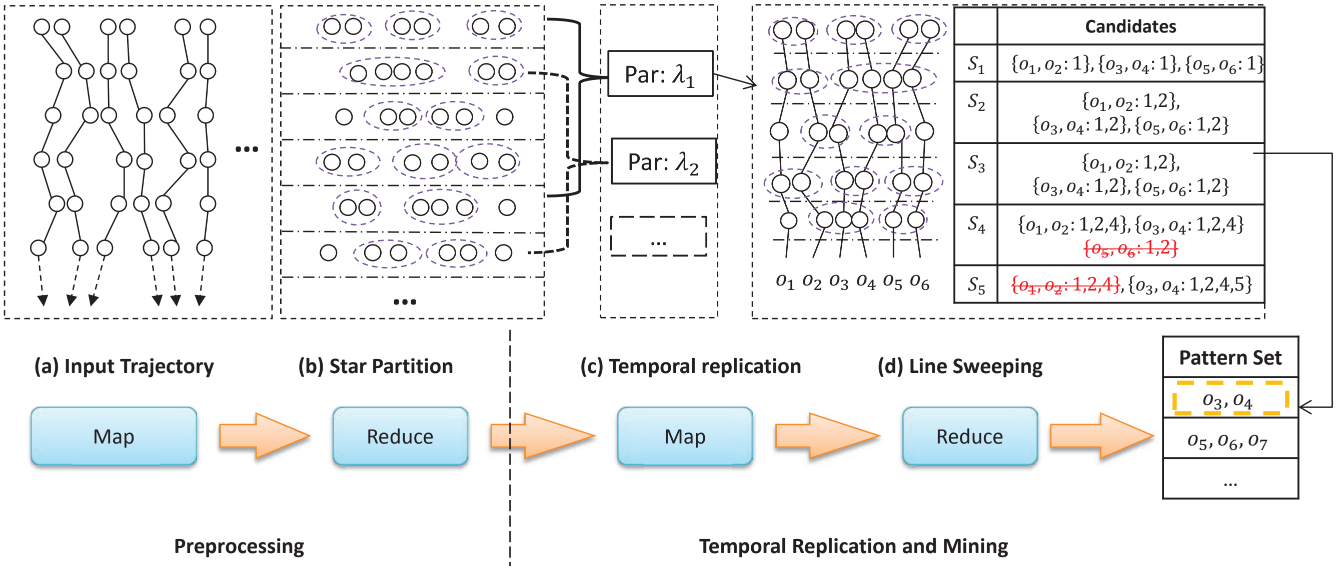

...
...
Par: λ_1
Par: λ_2
...
o_1 o_2 o_3 o_4 o_5 o_6
Candidates
S_1 $\{o_1, o_2: 1\}, \{o_3, o_4: 1\}, \{o_5, o_6: 1\}$
S_2 $\{o_1, o_2: 1,2\}, \{o_3, o_4: 1,2\}, \{o_5, o_6: 1,2\}$
S_3 $\{o_1, o_2: 1,2\}, \{o_3, o_4: 1,2\}, \{o_5, o_6: 1,2\}$
S_4 $\{o_1, o_2: 1,2,4\}, \{o_3, o_4: 1,2,4\}$ ~~$\{o_5, o_6: 1,2\}$~~
S_5 ~~$\{o_1, o_2: 1,2,4\}$~~, $\{o_3, o_4: 1,2,4,5\}$
(a) Input Trajectory
(b) Star Partition
(c) Temporal replication
(d) Line Sweeping
Map
Reduce
Map
Reduce
Pattern Set
o_3, o_4
o_5, o_6, o_7
...
Preprocessing
Temporal Replication and Mining

Figure 4: Work flow of Temporal Replication and Mining. (a)(b) correspond to the first map-reduce cycle which clusters objects in each snapshot; (c)(d) correspond to the second map-reduce cycle which uses temporal replication to mine GCMP in parallel.

C is initialized by inserting clusters at S_t (lines 2-4). During scanning snapshot S_j, candidates in C are joined with clusters at S_j. In the join, a candidate grows its temporal sequence while potentially reduces its object set. After the join, false patterns are deleted (line 7). Note that the size of C is always decreasing, therefore the complexity of LSM λ_t is $O(\eta|S_t||\overline{S}|)$,where $|\overline{S}|$ is the average snapshot size in λ_t.

Algorithm 1 Line Sweep Mining

```
Require: $\lambda_t = \{S_t, ..., S_{t+\eta-1}\}$
1: $C \leftarrow \{\}$                              ▷ Candidate set
2: for $c \in S_t$ do
3:     $C$.add($\langle c, t \rangle$)
4: end for
5: for all $j \in [t+1, t+\eta-1]$ do
6:     $C \leftarrow S_j \oplus C$
7:     remove false candidates from $C$
8: end for
9: output true candidates in $C$
```

The complete picture of temporal replication and parallel mining is summarized in Algorithm 2. We illustrate the workflow of TRPM method using Figure 4 (c)(d) with pattern parameters $M = 2, K = 3, L = 2, G = 2$. By Theorem 1, η is calculated as $(\lceil \frac{K}{L} \rceil - 1) * (G - 1) + 2K - 2 = 5$. Therefore, in Figure 4 (c), every 5 consecutive snapshots are combined into a partition in the map phase. In Figure 4 (d), a line sweep method is illustrated for partition λ_1. Let C_i be the candidate set during sweeping snapshot S_i. Initially, C_1 contains patterns whose object set is in snapshot S_1. As line sweeps, the patterns in C_i grows. At snapshot S_4, the candidate $\{o_5, o_6\}$ is removed. This is because the gap between its latest timestamp (i.e., 2) and the next scanning timestamp (i.e., 5) is 3, which violates the G constraint. Next, at snapshot S_5, the candidate $\{o_1, o_2\}$ is removed. This is because its local consecutive timestamps $\{4\}$ has only size 1, which violates the L constraint. Finally, $\{o_3, o_4\}$ is the qualified pattern and is outputted. Note that the minimum η under this setting is 5. If η is chosen as 4, the pattern $\{o_3, o_4\}$ would be excluded.

Algorithm 2 Temporal Replication and Parallel Mining

```
Require: list of $\langle t, S_t \rangle$ pairs
$\eta \leftarrow (\lceil \frac{K}{L} \rceil - 1) * (G - 1) + K + L - 1$
—Map Phase—
for all $\langle t, S_t \rangle$ do
    for all $i \in 1...\eta - 1$ do
        emit a $\langle \max(t - i, 0), S_t \rangle$ pair
    end for
end for
—Partition and Shuffle Phase—
for all $\langle t, S \rangle$ pair do
    group-by $t$, emit a $\langle t, \lambda_t \rangle$
    where $\lambda_t = \{S_t, S_{t+1}, ..S_{t+\eta-1}\}$
end for
—Reduce Phase—
for all $\langle t, \lambda_t \rangle$ do
    lineSweepMining($\lambda_t$)
end for
```

5. SPARE: STAR PARTITIONING AND APRIORI ENUMERATOR

The aforementioned replicate partitioning is based on the temporal dimension and results in groups of snapshots in each partition. ADD MORE DETAILS ABOUT THE LIMITATIONS HERE. To resolve the limitations caused by the replicate partitioning, we propose a new Star Partitioning and ApRiori Enumerator, named SPARE, to replace the second cycle of map-reduce jobs in Figure 4. Our new parallel mining framework is shown in Figure 5. Its input is the

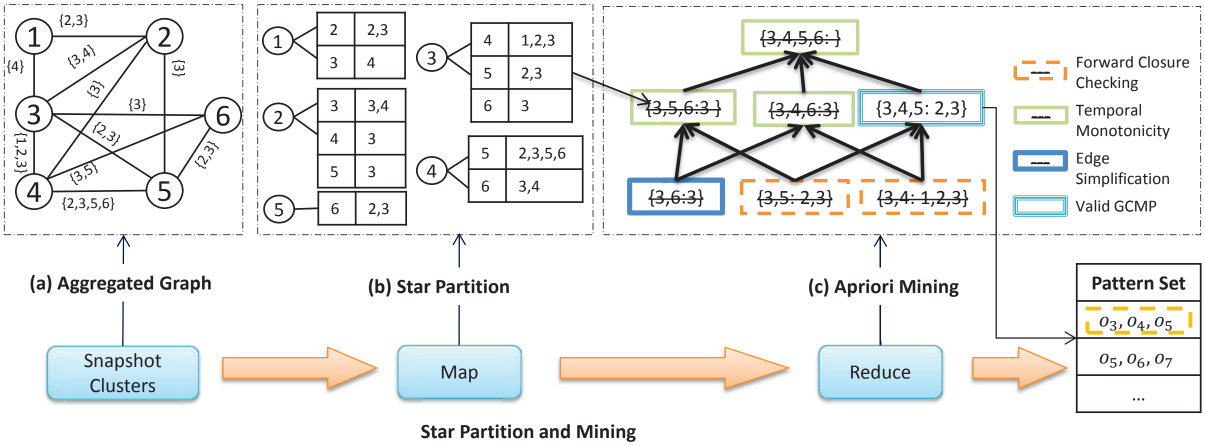


Figure 5: Star partition and mining. (a) Conceptual connection graph from Figure 1.(b) Five star partitions are generated (c) Apriori Mining with various pruning techniques.

set of clusters generated in each snapshot and the output contains all the valid GCMP patterns. In the following, we explain the two major components: star partitioning and apriori enumerator.

5.1 Star Partitioning

Let G_t be a graph for snapshot S_t, in which each node is a moving object and two objects are connected if they appear in the same cluster. It is obvious that G_t consists of a set of small cliques. Based on G_t, we define an aggregated graph G_A to summarize the cluster relationship among all the snapshots. In G_A, two objects form an edge if they are connected in any G_ts. Furthermore, we attach an inverted list for each edge, storing the associated timestamps in which the two objects are connected. An example of G_A, built on the trajectory database in Figure 1, is shown in Figure 5 (a). As long as two objects are clustered in any timestamp, they are connected in G_A. The object pair (o_1, o_2) appears in two clusters at timestamps 2 and 3 and is thus associated with an inverted list $\{2, 3\}$.

We use *star* as the data structure to capture the pair relationships. To avoid duplication, as G_t is an undirected graph and an edge may appear in multiple stars, we enforce a global ordering among the objects and propose a concept named *directed star*.

Definition 4 (Directed Star)**.** *Given a vertex with global id s, its directed star Sr_s is defined as the set of neighboring vertices with global id $t > s$. We call s the star ID.*

With the global ordering, we can guarantee that each edge is contained in a unique star partition. Given the aggregated graph G_A in Figure 5 (a), we enumerate all the possible directed stars in Figure 5 (b). These stars are emitted from mappers to different reducers. The key is the star ID and the value is the neighbors in the star as well as the associated inverted lists. The reducer will then call the Apriori-based algorithm to enumerate all the valid GCMP patterns.

Before we introduce the Apriori enumerator, we are interested to examine the issue of global ordering on the moving objects. This is because assigning different IDs to the objects will result in different star partitioning results, which will eventually affect the workload balance among the map-reduce jobs. The job incurring performance bottleneck is often known as *straggler* [16, 17, 18]. In the context of star partitioning, a straggler refers to the job assigned with the maximum star partition. We use Γ to denote the size of a partition and Γ is set to the number of edges in a directed star[1]. It is straightforward that a star partitioning with small Γ is preferred. For example, Figure 6 gives two star partitioning results under different vertex ordering on the same graph. The top one has $\Gamma = 5$ while the bottom one has $\Gamma = 3$. Obviously, the bottom one with smaller Γ is much more balanced.

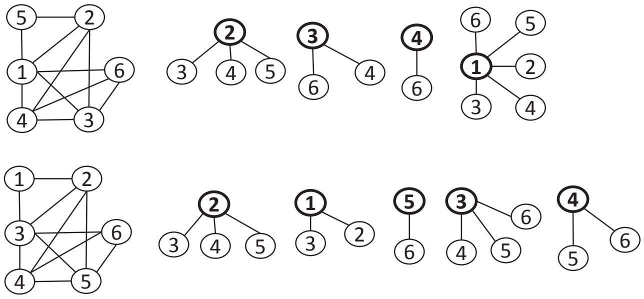


Figure 6: Examples of star partitioning with different vertex ordering.

Although it is very challenging to find the optimal vertex id ordering from the $n!$ possibilities, we observe that a random order can actually achieve satisfactory performance based on the following theorem.

Theorem 2. *Let Γ^* be the value derived from the optimal vertex ordering and Γ be value derived from a random vertex ordering. With probability $1 - 1/n$, we have $\Gamma = \Gamma^* + O(\sqrt{n \log n})$.*

Proof. In Appendix XXX. □

If G_A is a dense graph, we can get a tighter bound for $(\Gamma - \Gamma^*)$.

[1] A star is essentially a tree structure and the number of nodes equals the number of edges minus one.

Theorem 3. *Let d be the average degree in G_A. If $d \geq \sqrt{12 \log n}$, with high probability $1-1/n$, $\Gamma = \Gamma^* + O(\sqrt{d \log n})$.*

Proof. In Appendix XXXX. □

Hence, we can simply use object id to determine the vertex ordering in our implementation.

5.2 Apriori Enumerator

Intuitively, given a GCMP pattern with an object set $\{o_1, o_2, \ldots, o_m\}$, all the pairs of (o_i, o_j) with $1 \leq i < j \leq m$ must be connected in the associated temporal graphs $\{G_t\}$. This inspires us to leverage the classic Apriori algorithm to enumerate all the valid GCMP patterns starting from pairs of objects. However, we observe that the monotonicity property does not hold between an object set and its supersets.

Example 1. *In this example, we show that if an object set is not a valid pattern, we cannot prune all its super sets. Consider two candidates $P_1 = \{o_1, o_2 : 1, 2, 3, 6\}$ and $P_2 = \{o_1, o_3 : 1, 2, 3, 7\}$. Let $L = 2, K = 3$ and $G = 2$. Both candidates are not valid patterns because the constraint on L is not satisfied. However, when considering their object superset $\{o_1, o_2, o_3\}$, we can infer that their co-clustering timestamps are in $(1, 2, 3)$. This is a valid pattern conforming to the constraints of L, K, G. Thus, we need a new type of monotonicity to facilitate pruning.*

5.2.1 Monotonicity

To ensure the monotonicity, we first introduce a procedure named *star simplification*, to reduce the number of edges as well as unnecessary timestamps in the inverted lists. For instance, if the size of the inverted list for an edge e is smaller than K, then the edge can be safely removed because the number of timestamps in which its supersets are clustered must also be smaller than K. To generalize the idea, we propose three concepts named *maximal G-connected subsequence*, *decomposable sequence* and *sequence simplification*.

Definition 5 (Maximal G-connected Subsequence)**.** *A sequence T' is said to be a maximal G-connected subsequence of T if (1) T' is the subsequence of T, i.e., $\exists i \leq j, T' = T(i, \ldots, j)$, (2) T' is G-connected, and (3) there exists no other subsequence T'' of T such that T' is the subsequence of T'' and T'' is G-connected.*

Example 2. *Suppose $G = 2$ and consider two sequences $T_1 = (1, 2, 4, 5, 6, 9, 10, 11, 13)$ and $T_2 = (1, 2, 4, 5, 6, 8, 9)$. T_1 has two maximal 2-connected subsequences:$T_1^A = (1, 2, 4, 5, 6)$ and $T_1^B = (9, 10, 11, 13)$. This is because the gap between T_1^A and T_1^B is 3 and it is impossible for the timestamps from T_1^A and T_1^B to form a new subsequence with $G \leq 2$. Since T_2 is 2-connected, T_2 has only one maximal 2-connected subsequence which is itself.*

The maximal G-connected subsequence has the following two properties:

Lemma 4. *Suppose $\{T_1, T_2, \ldots, T_m\}$ is the set of all maximal G-connected subsequences of T, we have (1) $T_i \cap T_j = \emptyset$ for $i \neq j$ and (2) $T_1 \cup T_2 \cup \ldots \cup T_m = T$.*

Proof. We assume $T_i \cap T_j \neq \emptyset$. Let $T_i = (T_i[1], T_i[2], \ldots, T_i[p])$ and $T_j = (T_j[1], T_j[2], \ldots, T_j[n])$. Suppose $T[x]$ is a timestamp occurring in both T_i and T_j. Let $T[y] = \min\{T_i[1], T_j[1]\}$, i.e., the minimum timestamp of $T_i[1]$ and $T_j[1]$ occurs at the y-th position of sequence T. Similarly, we assume $T[z] = \max\{T_i[p], T_j[n]\}$. Apparently, the two subsequences $T[y : x]$ and $T[x : z]$ are G-connected because T_i and T_j are both G-connected. Then, sequence $(T_y, \ldots, T_x, \ldots, T_z)$, the superset of T_i and T_j, is also G-connected. This contradicts with the assumptions that T_i and T_j are maximal G-connected subsequences.

To prove (2), we assume $\cup_i T_i$ does not cover all the timestamps in T. Then, we can find a subsequence $T' = T[x : x+t]$ such that $T[x-1] \in T_a$ $(1 \leq a \leq m)$, $T[x+t+1] \in T_b$ $(1 \leq b \leq m)$ and all the timestamps in T' is not included in any T_i. Let $g' = \min\{T[x]-T[x-1], T[x+t+1]-T[x+t]\}$. If $g' \leq G$, then it is easy to infer that T_a or T_b is not a maximal G-connected subsequence because we can combine it with $T[x]$ or $T[x+t]$ to a form superset which is also G-connected. If $g' > G$, T' itself is a maximal G-connected subsequence which is missed in $\cup T_i$. Both cases lead to contradiction. □

Lemma 5. *If T_1 is a subset of T_2, then for any maximal G-connected subsequence T_1' of T_1, we can find a maximal G-connected subsequence T_2' of T_2 such that T_1' is a subset of T_2'.*

Proof. Since $T_1' \subseteq T_1 \subseteq T_2$, we know T_1' is a G-connected subsequence of T_2. Based on Lemma 4, we can find a maximal G-connected subsequence of T_2, denoted by T_2', such that $T_1' \cap T_2' \neq \emptyset$. If there exists a timestamp $T_1'[x]$ such that $T_1'[x] \notin T_2'$, similar to the proof of case (1) in Lemma 4, we can obtain a contradiction. Thus, all the timestamps in T_1' must occur in T_2'. □

Definition 6 (Decomposable Sequence)**.** *T is decomposable if for any of its maximal G-connected subsequence T', we have (1) T' is L-consecutive; and (2) $|T'| \geq K$.*

Example 3. *Let $L = 2, K = 4$ and we follow the above example. T_1 is not a decomposable sequence because one of its maximal 2-connected subsequence (i.e., T_1^B) is not 2-consecutive. In contrast, T_2 is a decomposable sequence because the sequence itself is the maximal 2-connected subsequence, which is also 2-consecutive and with size no smaller than than 4.*

Definition 7 (Sequence Simplification)**.** *Given a sequence T, the simplification procedure $\mathtt{sim}(T) = g_{G,K} \cdot f_L(T)$ can be seen as a composite function with two steps:*

1. *f-step: remove segments of T that are not L-consecutive;*
2. *g-step: among the maximal G-connected subsequences of $f_L(T)$, remove those with size smaller than K.*

Example 4. *Take $T = (1, 2, 4, 5, 6, 9, 10, 11, 13)$ as an example for sequence simplification. Let $L = 2, K = 4$ and $G = 2$. In the f-step, T is reduced to $f_2(T) = (1, 2, 4, 5, 6, 9, 10, 11)$. The segment (13) is removed due to the constraint of $L = 2$. $f_2(T)$ has two maximal 2-consecutive subsequences: $(1, 2, 4, 5, 6)$ and $(9, 10, 11)$. Since $K = 4$, we will remove $(9, 10, 11)$ in the g-step. Finally, the output is $\mathtt{sim}(T) = (1, 2, 4, 5, 6)$.*

It is possible that the simplified sequence $\mathtt{sim}(T) = \emptyset$. For example, Let $T = (1, 2, 5, 6)$ and $L = 3$. All the segments will be removed in the f-step and the output is $\emptyset$. We define $\emptyset$ to be not decomposable. We provide an important property of the sequence simplification process as follows:

Lemma 6. *If sequence T is a superset of any decomposable sequence, then* $\texttt{sim}(T) \neq \emptyset$*.*

Proof. It is obvious that $\texttt{sim}(T)$ is a one-to-one function. Given an input sequence T, there is a unique $\texttt{sim}(T)$. Let T_p be a decomposable subset of T and we prove the lemma by showing that $\texttt{sim}(T)$ is a superset of T_p.

Suppose T_p can be decomposed into a set of maximal G-connected subsequences $T_p^1, \ldots, T_p^m$ $(m \geq 1)$. Since T_p is a subset of T, all the T_p^i are also subsets of T. By definition, each T_p^i is L-consecutive. Thus, in the f-step of $\texttt{sim}(T)$, none of T_p^i will be removed. In the g-step, based on Lemma 5, we know that each T_p^i has a superset in the maximal G-connected subsequences of $f_L(T)$. Since $|T_p^i| \geq K$, none of T_p^i will be removed in the g-step. Therefore, all the T_p^i will be retained after the simplification process and $\texttt{sim}(T) \neq \emptyset$. □

With Lemma 6, we are ready to define the *monotonicity* based on the simplified sequences to facilitate the pruning in the Apriori algorithm.

Theorem 7. *Given a candidate pattern $P = \{O : T\}$, if* $\texttt{sim}(P.T) = \emptyset$*, then any pattern candidate P' with $P.O \subseteq P'.O$ can be pruned.*

Proof. We prove by contradiction. Suppose there exists a valid pattern P_2 such that $P_2.O \supseteq P.O$. It is obvious that $P_2.T \subseteq P.T$. Based on the Definition 2, the following conditions hold: (1) $P_2.T$ is G-connected. (2) $|P_2.T| \geq K$ and (3) $P_2.T$ is L-consecutive. Note that the entire $P_2.T$ is G-connected. Thus, $P_2.T$ itself is the only maximal G-connected subsequence. Based on conditions (1),(2),(3) and Definition 6, $P_2.T$ is decomposable. Then, based on Lemma 6, we know $\texttt{sim}(T) \neq \emptyset$ because $P_2.T \subseteq P.T$ and $P_2.T$ is decomposable. This leads to a contradiction with $\texttt{sim}(P.T) = \emptyset$. □

5.2.2 Apriori Enumeration

We then design the Apriori enumeration method to systematically discover all valid patterns in a star. During the enumeration, we refer a candidate pattern P as a R-candidate if $|P.O| = R$. The enumeration method discover patterns in "bottom-up" order: in iteration i, all $(i+1)$-candidates grows to $(i+2)$-candidates. Enumeration ends if no further candidates can be generated.

The process of enumeration is shown in Algorithm 3. As a start, each edge is treated as a 2-candidate pattern, with the inverted list as the time sequence. These 2-candidates are kept in $\mathbf{G}$ as the ground set. During the algorithm, we maintain a set of candidates $\mathbf{C}$. $\mathbf{C}$ is initialized as $\mathbf{G}$.

As a start, each edge is treated as

To systematically discover valid patterns in each star,

we enumerate all valid patterns in "bottom-up" order: pattern candidates with smallest object sizes are enumerated first and grew to pattern candidates with larger object sizes. During the enumeration, we call a candidate pattern P a R-pattern, if $|P.O| = R$. The enumerator takes a star as input, and treat each edge as a 2-pattern.

The input of the enumerator is a star. Each edge in the star is treated a 2-candidate, with the inverted list to be the time sequence.

we design the *Apriori Mining* algorithm. We borrow the terminologies in classic Aprirori algorithm: a candidate pattern P is called R-pattern if the $|P.O| = R$. In Aprirori, candidates grows from 2-pattern level by level.

To describe the algorithm, we call a candidate pattern R-pattern if the size of its object set is R. Therefore, each edge in the star is effectively a 2-pattern. The intuition of Apriori mining is the observation that $(R+1)$-patterns can be generated from R-patterns and 2 patterns. Thus, we may iteratively enumerate pattern candidates with all possible sizes. In particular, initially, for each $e(s,v) = ET$, pattern $p = (\{s,v\}, ET)$ is formed. During each iteration, we generate $(R+1)$-patterns by joining R-patterns with the 2-patterns. Technically, the join between $p_1 = (O_1 : T_1)$ and $p_2 = (O_2 : T_2)$ generates a new pattern $p_3 = (O_1 \cup O_2 : T_1 \cap T_2)$. Note that in Sr_s, each R-pattern contains the object s, thus the join only grow a R-pattern at most to a $(R+1)$-pattern. Our mining algorithm stops where no further patterns are generated. The algorithm is illustrated as in Algorithm 3.

Algorithm 3 Apriori Mining

```
Require: $Sr_s$
$\mathbf{C} \leftarrow \{\}$, $\mathbf{G} \leftarrow \{\}$, $\mathbf{OT} \leftarrow \{\}$
for all $e(s,t) = T \in Sr_s$ do
    $T' \leftarrow \texttt{sim}(e(s,t))$
    $\mathbf{G}.\texttt{add}(\langle\{s,t\} : T'\rangle)$, unless $T' \neq \emptyset$
end for
$\mathbf{C} \leftarrow \mathbf{G}$
while $\mathbf{C} \neq \emptyset$ do
    $\mathbf{N} \leftarrow \{\}$, $U \leftarrow \{\}$
    for all $(c, g) \in \mathbf{C}$ do
        for all $g \in \mathbf{G}$ do
            $c' \leftarrow \langle c.O \cup g.O : \texttt{sim}(c.T \cap g.T)\rangle$
            $\mathbf{N}.add(c')$, unless $c'.T \neq \emptyset$
        end for
        $\mathbf{OT}.\texttt{add}(c)$, if $c$ is not extended.
    end for
    $U \leftarrow$ union of $n_i \in \mathbf{N}$
    for all $n \in \mathbf{N}$ do
        $U.O \leftarrow$, $U.T \leftarrow U.T \cap n.T$
    end for
    if $U$ is valid then
        Output.add($U$)
        break;
    end if
    $\mathbf{C} \leftarrow \mathbf{N}$
    Lv ← Lv ⊕ Ground                          ▷ Simplification
    Remove false patterns in Lv               ▷ Monotonicity
    if union of Lv is a valid pattern then    ▷ Forward
 Closure
        output.add(Lv)
        break
    end if
end while
output.addAll($Lv$)
return output
```

An illustration of Algorithm 3 is shown in Figure 5 (c). As shown, the star $Sr_3 = \{3, 4, 5, 6\}$ initially generate three 2-candidates. At every iteration, higher level candidates are generated by joining lower level candidates. When no more candidates can be generated, the algorithm stops by outputting the valid patterns.

EXPLAIN CLEARLY HOW YOU IMPLEMENT THE FORWARD CLOSURE CHECKING! BETTER SHOW IT IN THE ABOVE PSEDUDO CODE IN THE APRIORI ALGORITHM.

Although leveraging *temporal monotonicity* could largely prune false candidates and reduce the apriori search space, it is ineffective when a *true* pattern exists. In an extreme case, if a final pattern of a star Sr_s is the *union of all vertices* in the star, then in apriori, $\binom{|Sr_s|}{i+1}$ candidates needs to be generated at each level i. This results in an exponential search space while the output only contains one pattern. Generally, when candidates at level i collectively forms a true pattern, running aprior produces many wasted candidates.

Let Lv_i be the set of candidates at level i of Algorithm 3, we use the *forward closure* FC_i to denote the union of the objects in all candidates in Lv_i. Then, the *forward closure checking* is stated as follows:

Theorem 8 (Forward Closure Checking)**.** *Let Lv_i be the candidates generated at level i in Algorithm 3, if FC_i is a proper pattern, then it is safe to terminate Algorithm 3 and directly output FC_i.*

It is notable that, as the level grows in Algorithm 3, the closure FC reduces, thus the terminating power of FC would be stronger. We give hybrid example of the three optimization as follows:

Example 5. *We use Figure 5 (c) to demonstrate the power of candidate pruning. As shown, at the initial stage,* $\{3, 6 : 3\}$ *is first pruned by Edge Simplification since its timestamps fails to be a candidate sequence. Subsequently, all further candidates containing* $\{3, 6\}$ *are pruned by Monotonicity. Then, we check the Forward Closure of remaining candidates (i.e.,* $\{3, 4\}$ *and* $\{3, 5\}$*) and find* $\{3, 4, 5\}$ *is a valid candidate. Therefore,* $\{3, 4\}$ *and* $\{3, 5\}$ *are pruned, and* $\{3, 4, 5\}$ *is the output.*

5.3 Correctness of SPARE

We summarize the workflow of SPARE framework in Figure 5 as follows. After the parallel clustering in each snapshot, we build an aggregated graph G_A to capture the clustering relationship. In the map phrase, we partition G_A into stars and emit the star partitions to the reducers. Each reducer is an Apriori Enumerator. When receiving a star Sr_i, the reducer creates initial candidate patterns. Specifically, for each $o \in Sr_i$, a pattern candidate $\{o, i : e(o, i)\}$ is created. Then the Apriori algorithm enumerate the true patterns from the candidate patterns. Meanwhile, the *simplification*, *monotonicity pruning* and *forward closure checking* are applied to boost the enumeration process.

In the following, we show the correctness of the SPARE framework.

Theorem 9. *The SPARE framework guarantees completeness and soundness.*

Proof. If P is a valid pattern in original trajectories, let s be the object with smallest ID in $P.O$. Based on the definition of GCMP, $\forall t \in P.T$, $\forall o \in P.O$, $C_t(s) = C_t(o)$. It follows that all object $o \in P.O$ are in Sr_s. Furthermore, every timestamp in $P.T$ is included in Sr_s. Therefore, P is a valid pattern in Sr_s. This ensures the *completeness*. For any pattern P enumerated by SPARE, by definition of star partition, $\forall o_1, o_2 \in P.O$, $\forall t \in P.T$, $C_t(o_1) = C_t(o_2)$. This ensure the *soundness*. □

.

In fact, besides that every valid patterns can be mined in some stars, star partition further ensures that patterns discovered from different star partitions cannot be the same. This can be formulated as:

Lemma 10 (Pattern Uniqueness of Star Partition)**.** *Let Sr_i and Sr_j $(i \neq j)$ be two star partitions. Let P_i (resp. P_j) be the patterns discovered from Sr_i (resp. Sr_j). Then, $\forall p_i \in P_i, \forall p_j \in P_j$, we have $p_i.O \not\equiv p_j.O$.*

Proof. The correctness of Lemma 10 can be directly derived from the above arguments: let o_i be the smallest element in $p_i.O$ and o_j be the smallest element in $p_j.O$. Since p_i and p_j belongs to different stars, then $o_i \neq o_j$, therefore $p_i.O \not\equiv p_j.O$. This proves the lemma. □

This lemma verifies the superiority of the SPARE framework over the TRPM framework.

6. EXPERIMENTAL STUDY

In this section, we present our experimental findings on deploying our GCMP detectors to large-scaled real trajectories. All algorithms are were implemented in Java 1.7., and all the experiments are carried out on our in-house cluster: Dianwei. The cluster includes 12 nodes, each of which has four quad-core 2.2GHz Intel processor, 32 GB memory and gigabit Ethernet. The cluster is installed with CentOS 5.5. operating system, Java 1.7.0 VM.

Experiment Setup: We use Yarn[2] to manage the cluster. One node is dedicated as Yarn's master node, and each other node reserves one core and 2 GB memory for Yarn processes. We use Apache Spark [**?**] as the MapReduce framework. Spark takes the remaining 11 nodes as the computing nodes. To fully utilize the computing power of the cluster, we assign each node to run five executors, each executor takes three cores and 5 GB memory. We use "yarn-cluster" mode for Spark which randomly picks one executor to act as the Application Master. Such a configuration allows us to run 162 tasks at the same time. We use HDFS as our storage engine with replicator factor of 1. The configurations of important parameters are summaries in the table below:

Parameter	Value
Java Version	1.7.0
spark.driver.memory	2GB
spark.executor.cores	3
spark.executor.instances	54
spark.executor.memory	5GB
spark.master	yarn-cluster
spark.serializer	KryoSerializer
spark.shuffle.spill	true

Datasets: We prepare three real datasets for study. The details of the datasets are as follows:

[2] http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

- Geolife [3]: this dataset collects 18,670 trajectories for passengers in Beijing over three years. The data are collected per around 5 seconds. Each data point records a trajectory ID and the latitude/longitude information.

- Shopping [4]: this dataset contains visitors trajectories in ATC shopping center in Osaka. The samples are taken per around 0.5 seconds. There are in total 13,183 trajectories. Each data point is a trajectory ID with in-door coordinates.

- Taxi: this dataset records 15,054 Singapore taxi trajectories over one month span. The sample rate is around 30 seconds. Each data point is the taxi plate with latitude/longitude.

Preprocessing: We replace timestamps with a global sequence for each dataset. The sequence number 0 is the earliest timestamp among all trajectories in a dataset. We use the sampling rate of each dataset as the tick of the sequence number and every data point is mapped to the nearest ticks. If several points mapped to the same tick, they are merged by averaging the location coordinates. For missing data within small intervals (i.e., sequence difference less than 10), we use linear interpolation to fill them. We then use DBSCAN ($\epsilon = 5, minPt = 10$ for GeoLife and Shopping, $\epsilon = 20, minPt = 10$ for Taxi) as the clustering algorithm for preprocessing. Note that both our TRPM and SPARSE algorithms treat the preprocessing as a black box and other spatial clustering methods are also applicable. The clustered snapshots are stored in HDFS in $\langle t, S_t \rangle$ pair, where t is the timestamp, S_t contains the clusters at snapshot t. After preprocessing, the summary of statistics are presented as in Table 4.

Attributes	**Geolife**	**ACTShopping**	**SingTaxi**
# objects	18,670	13,183	15,054
# average ts	2,924	3,114	19,667
# data points	54,594,696	41,052,242	296,075,837
# snapshots	10,699	16,931	44,364
# clusters	206,704	211,403	536,804
avg. cluster size	223	171	484
# of patterns	441	369	585

Table 4: Statistics of data set

Parameters: We study the performance of the GCMP detectors under various conditions. The variables to be tested and their value ranges are listed in Table 5. The default values are highlighted in bold.

Variables	**Meaning**	**Values**
M	min size of object set	5, 10, **15**, 20, 25
K	min duration	120, 150, **180**, 210, 240
L	min local duration	10, 20, **30**, 40,50
G	max gap	10, 15, **20**, 25, 30
N	number of executors	1, 14, 24, 34, 44, **54**

Table 5: Variables and their default values during experiments.

6.1 Effects of G on GCMP

We first study the effects of introducing gap parameter G on pattern discovery. As shown in Figure 3, existing works are not able to handle large trajectories, we sample and cluster 1 million trajectory points from the three real data. We use two instances of GCMPs, GCMP-L where the L value is default and GCMP where $L = 1$. We measure the number of patterns discvoered by different methods. The results are presented in Figure 7.

We can see from Figures 7 that the number of patterns discovered by GCMP is positively related to G. When $G = 1$, the number of patterns returned by GCMP and GCMP-L are identical to *convoy*. As G grows, the number returned by GCMP-L converges to *platoon*. For example, in (a), two patterns converge when G is 40. Similarly in (b) and (c), two patterns converge when G is 50. The reason is that when G is large enough, all *platoon*s are G-connected. Similar trend applies between GCMP and *swarm*. As see from the figures, in (a), two patterns converge at $G = 50$; while in (b) and (c) the two patterns converge at $G = 60$ and $G = 40$ respectively. We further note that, existing patterns has different granularity in temporal domain. As we seen from the three figures, the number of *swarm* is greater than *platoon* and *convoy*. Despite none of the existing patterns are able to fine-grain control the temporal domain of patterns, GCMP can discover the patterns more precisely. These results confirms the usefulness of GCMP in expressing other patterns.

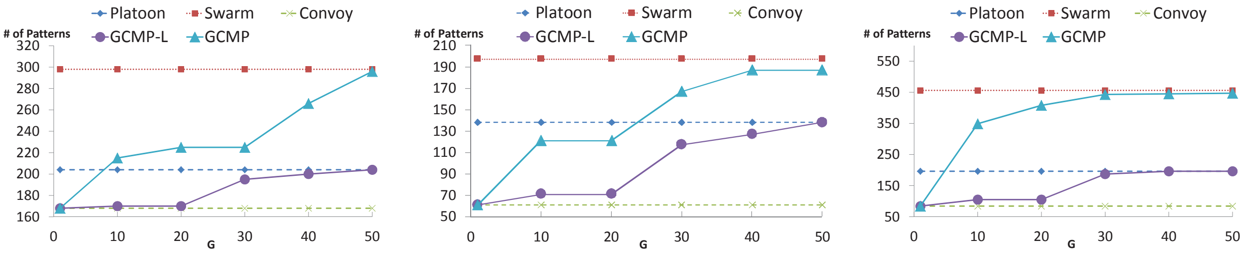
6.2 Performance Comparison

To study the performance of TRMP and SPARE, we run both algorithms in all three datasets under different parameter settings. We report their overall performance as in Figures 8. In brief, both algorithms are able to handle the large-scaled real datasets. However, we observe that TRMP's performance is very dependent on the pattern parameters. In cases when G is large, TRMP takes near three hours (e.g., in Taxi dataset). In contrast, SPARE performs quite stable. Another general observation is that both algorithms run slower in the Taxi dataset as compared to in other two datasets. This is because Taxi dataset contains the most number of temporal data points, which is around 7 times of Shopping dataset and 5 times of geolife dataset. We observe that, in almost all cases, SPARE outperforms TRMP and analyze the detail of each scenario separately:

Vary M: Figures 8 (a),(e),(i) present the performance of TRPM and SPARE when M changes under three datasets. As the figures show, SPARE runs much faster than TRPM. We can see that SPARE speeds up TRPM 2.7 times in Shopping data, 3.1 times in Geolife data and 7 times in Taxi data. Another observation is that, the running times of both algorithms slightly decrease as M grows. This is because when M becomes bigger, smaller clusters (stars) in the reduce phase of TRPM and SPARE can be directly pruned.

Vary K: Figure 8 (b),(f),(j) presents the performance of TRPM and SPARE when K changes under three datasets. There are two interesting findings in study the performance wrt. K. First, SPARE and TRPM takes different trends as K increases. Whe K increases, SPARE tends to have faster performances while TRPM's performances continuously slow down. For SPARE, the good performances result from more pruning powers brought by K. When K increases, many shorter time sequences in the reduce phase are pruned. In contrast, K does not bring much pruning

[3] http://research.microsoft.com/en-us/projects/geolife/
[4] http://www.irc.atr.jp/crest2010_HRI/ATC_dataset/



(a) Geolife (b) ACTShopping (c) SingTaxi

Figure 7: Effectiveness of GCMP model on sampled datasets.

power during the reduce phase of TRPM. Even K is very large, each reducers in TRPM still has to sweep the entire partition to determine whether a pattern exists. Moreover, η in TRPM grows linearly with K. This indicates that more data are shuffled and more snapshots are need to be swept in each partition. This explains the low performance of TRPM as K increases. Another interesting finding is that, when K is small, TRPM tends to outperform SPARE. This is because that when K is small, shuffle data of TRPM is comparable to SPARE. Meanwhile, small K indicates small size of partition in TRPM but low pruning power in SPARE. This explains the superiority of TRPM. Note that, when K is small, TRPM is at most 1.3 times faster than SPARE and the absolute time is less than three minutes.

Vary L:Figure **??** presents the performance of TRPM and SPARE when L changes under three datasets.

Vary G:Figure **??** presents the performance of TRPM and SPARE when G changes under three datasets.

6.2.1 *Effects of pattern parameters* M, L, K, G

6.3 SPARE Analysis

6.3.1 *Load Balance*

6.3.2 *Scalability*

7. CONCLUSION AND FUTURE WORK

8. REFERENCES

[1] Y. Zheng, Y. Liu, J. Yuan, and X. Xie, “Urban computing with taxicabs,” in *Proceedings of the 13th international conference on Ubiquitous computing*, pp. 89–98, ACM, 2011.

[2] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye, “Mining periodic behaviors for moving objects,” in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1099–1108, ACM, 2010.

[3] J. Bao, Y. Zheng, D. Wilkie, and M. F. Mokbel, “A survey on recommendations in locationbased social networks. submitted to,” *Geoinformatica*, 2013.

[4] X. Li, *Managing moving objects and their trajectories*. PhD thesis, National University of Singapore, 2013.

[5] J. Gudmundsson and M. van Kreveld, “Computing longest duration flocks in trajectory data,” in *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pp. 35–42, ACM, 2006.

[6] Y. Wang, E.-P. Lim, and S.-Y. Hwang, “Efficient mining of group patterns from user movement data,” *Data & Knowledge Engineering*, vol. 57, no. 3, pp. 240–282, 2006.

[7] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, “Discovery of convoys in trajectory databases,” *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1068–1080, 2008.

[8] Z. Li, B. Ding, J. Han, and R. Kays, “Swarm: Mining relaxed temporal moving object clusters,” *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 723–734, 2010.

[9] Y. Li, J. Bailey, and L. Kulik, “Efficient mining of platoon patterns in trajectory databases,” *Data & Knowledge Engineering*, 2015.

[10] J. Gudmundsson, M. van Kreveld, and B. Speckmann, “Efficient detection of motion patterns in spatio-temporal data sets,” in *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pp. 250–257, ACM, 2004.

[11] Y. Zheng, “Trajectory data mining: an overview,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, no. 3, p. 29, 2015.

[12] D. Birant and A. Kut, “St-dbscan: An algorithm for clustering spatial–temporal data,” *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.

[13] P. Laube, M. van Kreveld, and S. Imfeld, “Finding remodetecting relative motion patterns in geospatial lifelines,” in *Developments in spatial data handling*, pp. 201–215, Springer, 2005.

[14] D. H. Douglas and T. K. Peucker, “Algorithms for the reduction of the number of points required to represent a digitized line or its caricature,” *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973. doi:10.3138/FM57-6770-U75U-7727.

[15] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[16] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, “Skewtune: mitigating skew in mapreduce applications,” in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 25–36, ACM, 2012.

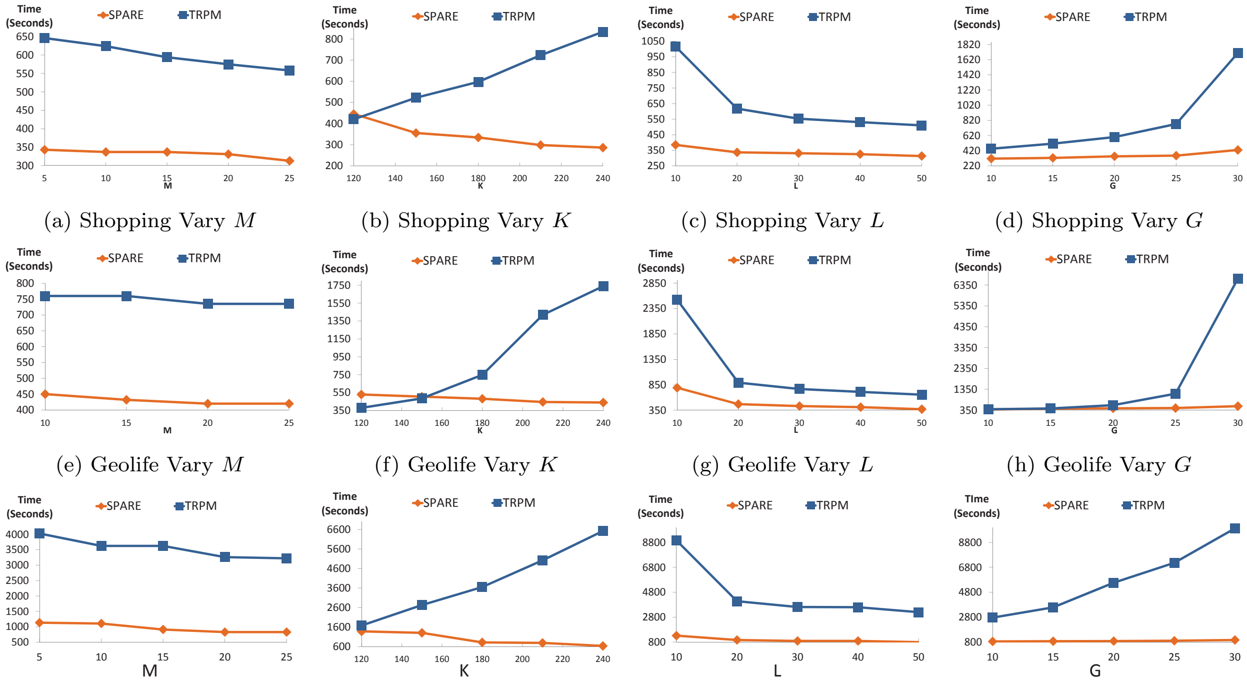
[17] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, “Shark: Sql and rich analytics at scale,” in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*, pp. 13–24, ACM, 2013.

[18] E. Coppa and I. Finocchi, “On data skewness, stragglers, and mapreduce progress indicators,” in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pp. 139–152, ACM, 2015.

APPENDIX

A. PROOFS OF THEOREMS

A.1 Proof of Theorem 1



(a) Shopping Vary M
(b) Shopping Vary K
(c) Shopping Vary L
(d) Shopping Vary G

(e) Geolife Vary M
(f) Geolife Vary K
(g) Geolife Vary L
(h) Geolife Vary G

(i) Taxi Vary M
(j) Taxi Vary K
(k) Taxi Vary L
(l) Taxi Vary G

Figure 8: Performance of SPARE and TRPM on real datasets under different pattern parameters.

Proof. We compute η by analyzing range($\rho_{G,L,K}(\cdot)$). In the following, when no ambiguity, we use $\rho(T)$ to represent $\rho_{G,L,K}(T)$. For any sequence T, its $\rho(T)$ can be viewed as interleaving segments and gaps. We may view $\rho(T)$ as $l_1, g_1, \dots, l_{n-1}, g_{n-1}, l_n$, where l_i is a segment and g_i is a gap. For simplicity, we use l_i and g_i directly as their sizes. Then the range of $\rho(T)$ is $\Sigma_{i=1}^{i=n} l_i + \Sigma_{i=1}^{i=n-1} g_i$. Since $\rho(T)$ is valid, the following constraints holds: (1) $\Sigma_{i=1}^{i=n} l_i \geq K$; (2) $\forall l_i,\ l_i \geq L$; (3) $\forall g_i,\ g_i \leq G-1$. Based on constraint (1) and (2), $n \leq \lceil \frac{K}{L} \rceil$. This indicates that $\Sigma_{i=1}^{i=n-1} g_i \leq (\lceil \frac{K}{L} \rceil - 1)(G-1)$. If $l_n > L$ and $\Sigma_{i=1}^{i=n} l_i > K$, we can reduce the range of $\rho(T)$ by reducing l_n. If $\Sigma_{i=1}^{i=n} l_i - K < l_n - L$ then, $\Sigma_{i=1}^{i=n} l_i - K$ can be reduced to K. In this case, the maximum range of $\rho(T)$ is $\leq K + (\lceil \frac{K}{L} \rceil - 1)(G-1)$. Otherwise, we can reduce l_n to be L. When $l_n = L$, since we wish $\rho(T)$ to be minimum, it indicates $\Sigma_{i=1}^{i=n} l_i \leq K-1$. Therefore $\Sigma_{i=1}^{i=n} l_i \leq K-1+L$. In this case, the maximum range of $\rho(T)$ is $K-1+L+(\lceil \frac{K}{L} \rceil - 1)(G-1)$. Taking maximum, the maximal possible $\rho(T)$ for all Ts is thus $K-1+L+(\lceil \frac{K}{L} \rceil - 1)(G-1)$. This proves the first half of the theorem.

For the second half of theorem, we prove by construction. Given G, L, K, let η^* be the optimal value. Then consider the sequence T generated by replicate the following pattern: a L-segment followed by G-gap. The replication stops when $|T| \geq K$. Apparently T is valid wrt. G, L, K. It is easy to see that the minimum range($\rho(T)$) is $(\lceil \frac{K}{L} \rceil - 1)(G-1) + K$. Since η^* is optimal, $\eta^* \geq$ range$(\rho(T)) = (\lceil \frac{K}{L} \rceil - 1)(G-1) + K$. Recall $\eta = (\lceil \frac{K}{L} \rceil - 1)(G-1) + K + L - 1$, this implies that $\eta \leq \eta^* + L - 1$. □

A.2 Proof of Theorem 2

Proof. We formalize optimization of Γ using a linear algebra model as follows: Let G_A be an aggregate graph, with a $n \times n$ adjacent matrix J. A vertex order is in fact a permutation of J. Therefore, the adjacent matrices of any reordered graphs can be represented as PJP^T where $P \in \mathbb{P}$ is a *permutation matrix* [5] with dimension n. Since in star partition, we assign each edge $e(i,j)$ in G_A to the lower vertex, then the matrix $B = \text{triu}(PJP^T)$ [6] represents the assignment matrix wrt. P (i.e., $b_{i,j} = 1$ if vertex j is in star Sr_i). Let vector $\vec{b}$ be the *one*[7] vector of size n. Let $\vec{c} = B\vec{b}$, then each c_i denotes the number of edges in star Sr_i. Thus, Γ can be represented as the infinity norm of $B\vec{b}$. Let Γ^* be the minimum Γ among all vertex orders. Γ^* can then be formulated as follows:

$$\Gamma^* = \min_{P \in \mathbb{P}} ||B\vec{b}||_\infty \text{ ,where } ||B\vec{b}||_\infty = \max_{1 \leq j \leq n} (c_j) \qquad (2)$$

Let $B*$ be the assignment matrix wrt the optimal vertex order. Since we have a star for each object, by the degree-sum formula and pigeon-hole theorem, $\Gamma^* = ||B^*\vec{b}||_\infty \geq d/2$. Next, given a numbering P, let $e_{i,j}$ be an entry in PAP^T. Since edges in graph G are independent, then $e_{i,j}$s are independent. Moreover, let d_i denote the degree of vertex i, then $E[d_i] = E[\Sigma_{1 \leq j \leq n} e_{i,j}] = d$. This is because renumbering the vertexes does not affect the average degree. Since $B = \text{triu}(PAP^T)$, entries in B can be written as :

$$b_{i,j} = \begin{cases} e_{i,j}, i > j \\ 0, otherwise \end{cases}$$

There are two observations made. First, since $e_{i,j}$s are independent, $b_{i,j}$s are independent. Second, since $i > j$ and $e_{i,j}$s are independent. $E[b_{i,j}] = E[e_{i,j}]E[i > j] = E[e_{i,j}]/2$.

By definition, $c_i = \Sigma_{1 \leq j \leq n} b_{i,j}$, is a sum of n independent 0-1 variables. Taking expectation on both sides, we get: $E[c_i] = E[\Sigma_{1 \leq j \leq n} b_{i,j}] = E[\Sigma_{1 \leq j \leq n} e_{i,j}]/2 = d/2$. Let $\mu = E[c_i] = d/2$, $t = \sqrt{n \log n}$, by Hoeffding's Inequality, the following holds:

$$Pr(c_i \geq \mu + t) \leq \exp(\frac{-2t^2}{n})$$
$$= \exp(-2 \log n) = n^{-2}$$

[5] an identity matrix with rows shuffled

[6] triu is the upper triangle part of a matrix

[7] every element in $\vec{b}$ is 1

The first step is due to the fact that all $b_{i,j}$ are bounded in the range of [0,1]. Next, since the event $(\max_{1\leq j\leq n}(c_j) \geq \mu + t)$ can be viewed as $\cup_{c_i}(c_i \geq \mu + t)$, by Union Bound, we achieve the following:

$$\begin{aligned} Pr(\Gamma \geq \mu + t) &= Pr(\max_{1\leq j\leq n}(c_j) \geq \mu + t) \\ &= Pr(\cup_{c_i}(c_i \geq \mu + t)) \\ &\leq \Sigma_{1\leq i\leq n} Pr(c_i \geq \mu + t) \\ &= n^{-1} = 1/n \end{aligned}$$

Substitute back t and μ, we achieve the following concise form:

$$Pr(\Gamma \geq (d/2 + \sqrt{n \log n})) \leq 1/n$$

This indicates that, the probability of $(\Gamma - d/2)$ being less than or equal to $O(\sqrt{n \log n})$ is $(1 - 1/n)$. With the fact that $\Gamma^* \geq d/2$, we conclude that with probability greater than $(1 - 1/n)$, the difference between Γ and Γ^* is less than $O(\sqrt{n \log n})$. When the aggregated graph is *dense* (i.e., $d \geq \sqrt{12 \log n}$), we may use the Chernoff Bound instead of Hoeffding's Inequality to derive a tighter bound of $O(\sqrt{\log n})$ with the similar reasoning. □

A.3 Proof of Theorem 8

Proof. We prove by contradiction. Suppose there exists another pattern P such that $P.O \neq FC_i$, let $X = P.O - FC_i \neq \emptyset$. Consider a subset P_1 of P which contains X with size $i + 1$, (i.e., $P_1 \subseteq P, P_1 \subseteq X, |P_1| = i + 1$). Since P is a proper pattern, then P_1 is also a proper pattern. Therefore $P_1 \in Lv_i$. It follows X is in the forward closure of FC_i, (i.e.,$X \in FC_i$), which contradicts with $X \notin FC_i$. □